UNIVERSIDADE DA BEIRA INTERIOR
Covilhã | Portugal

Sistema de Animação Multi-Resolução

Gonçalo da Cruz Moura Dias

Dissertação para obtenção do Grau Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor Frutuoso Gomes Mendes da Silva

Covilhã, Portugal
Outubro 2010

À família e amigos

Agradecimentos

A todos os que directa ou indirectamente tornaram possível a realização desta dissertação. Ao meu orientador, Prof. Doutor Frutuoso Gomes Mendes da Silva pelo apoio, confiança e motivação prestados.

Aos meus amigos Etelvina Pinho e Miguel Gaspar pelas discussões sobre esta dissertação e pelas ideias prestadas que permitiram resolver alguns problemas.

Aos meus amigos Rui Brás, Catarina Nunes e Gonçalo Amador pelo ânimo, compreensão, apoio e ajuda que me deram em algumas situações.

Às minhas amigas Irene Ferreira e Ana Rita pelos momentos de divertimento e descontracção que passámos juntos pois sem isso tornar-se-ia mais difícil a realização deste trabalho.

À Aurora (minha madrinha de baptismo) por me dar apoio e força em certos momentos.

À Ruanita (a minha cadela de estimação) por me "obrigar" a desanuviar em algumas alturas pois sem isso seria mais difícil a realização deste trabalho.

A todos os meus amigos e familiares que não são mencionados aqui mas que me deram forças e acreditaram em mim para que eu continuasse.

Em especial: ao meu **Pai**, à minha **Mãe**, à minha **Irmã** e ao **David** por tudo, pois sem eles nada disto seria possível.

Resumo

As malhas multi-resolução são cada vez mais utilizadas em computação gráfica, pois permitem ter várias resoluções de um mesmo modelo. Devido ao desenvolvimento de tecnologias para aquisição de dados 3D, como por exemplo, os scanners 3D e devido ao detalhe com que estes dispositivos conseguem criar as representações geométricas dos modelos 3D, é necessário muitas vezes recorrer a algoritmos de simplificação de malhas.

O trabalho desenvolvido nesta dissertação, teve como objectivo testar a viabilidade da aplicação de um esquema multi-resolução para malhas aplicado a um sistema de animação, ou seja, consoante a distância da malha ao observador, esta vai sendo simplificada ou refinada. Assim, quando a malha se afasta do observador é aplicado um algoritmo de simplificação de modo a gerar uma representação com menos detalhe, de modo inverso quando a malha se aproxima é gerada uma malha com mais detalhe através de um algoritmo de refinamento. Tal como na vida real, o detalhe que uma pessoa tem dos objectos depende da distância a que está deles, pois se o objecto estiver muito longe os seus pormenores não são todos perceptíveis, enquanto que se o objecto estiver perto do utilizador os pormenores já são perceptíveis.

Acrónimos

AIF Adjacency and Incidence Framework

BSP Bi-Star Planarity

CAD Computer Aided Design

EDCF Estrutura de Dados baseada na Célula Fantasma

FPS Frames Por Segundo

LOD Levels Of Detail

NSA Normal-based Simplification Algorithm

OFF Object File Format

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Neste capítulo, são apresentadas as motivações e os objectivos deste trabalho, assim como a estrutura da dissertação.

1.1 Motivações e Objectivos

A realização deste trabalho teve como principal objectivo avaliar a viabilidade da análise multi-resolução para representar modelos em sistemas de animação, ou seja, em sistemas de tempo real.

As malhas triangulares são as primitivas mais populares em Computação Gráfica, uma vez que existe suporte de *hardware* para a renderização de malhas triangulares e por isso vários dispositivos de *hardware* operam mais eficientemente em triângulos do que em outro tipo de primitivas. Este tipo de primitivas é utilizado por várias fontes de dados, tais como: digitalização laser, modelação digital de terrenos, modelos de *Computer Aided Design (CAD)*, geração de iso-superfícies.

Hoje em dia a análise multi-resolução é utilizada para representar malhas poligonais com diferentes resoluções. No entanto para objectos com poucas células (vértices, arestas e faces), exige pouca memória e o seu processamento é rápido, mas para objectos com muitas células exige mais memória e o processamento é mais lento [4]. Existem normalmente duas soluções para representar objectos com diferentes níveis de detalhe (isto é, com diferentes resoluções). A primeira são os modelos *Levels Of Detail (LOD)* e a segunda a análise multi-resolução que permite adaptar a resolução do modelo em tempo de execução (*runtime*).

No caso dos modelos LOD são mantidas em memória várias instâncias geométricas do mesmo objecto mas com resoluções diferentes o que faz com que exista um acréscimo de memória. No entanto, esta abordagem permite a rápida troca entre níveis de detalhe pois não exige processamento. O principal objectivo dos LOD é melhorar o desempenho quando se representam

os objectos com diferentes níveis de detalhe [5, 6].
No caso da análise multi-resolução é mantida uma única instância geométrica em memória e a passagem entre níveis de detalhe é feita através de algoritmos de simplificação e/ou refinamento, logo requer tempo na passagem entre níveis de detalhe. Os algoritmos de simplificação permitem simplificar a malha de forma a reduzir o nível de detalhe da malha. Os algoritmos de refinamento permitem refinar a malha simplificada, por exemplo, guardando informação durante a fase de simplificação da malha.
Geralmente a solução mais utilizada em aplicações de tempo real são os LOD; pois apesar de consumir mais memória tem a vantagem de tornar mais rápida a passagem entre níveis de detalhe. Assim, neste trabalho tentou-se avaliar a viabilidade da análise multi-resolução para representação de modelos, por exemplo, para jogos e animação.
O trabalho baseou-se no uso de malhas poligonais multi-resolução recorrendo a algoritmos de simplificação e refinamento de malhas. Nesta dissertação avaliou-se o uso de algoritmos de simplificação e refinamento de malhas, tendo em conta a dimensão das malhas para o possível uso em tempo real. Para isso desenvolveram-se os algoritmos de simplificação e refinamento de malhas e fizeram-se alguns testes para modelos de diferentes dimensões.

1.2 Estrutura da Dissertação

Esta dissertação é composta por quatro capítulos.

- Neste capítulo são apresentadas as motivações e objectivos do trabalho, bem como a estrutura da dissertação.
- No Capítulo 2 é efectuado um levantamento do trabalho relacionado no que respeita aos esquemas multi-resolução.
- No Capítulo 3 é apresentado o trabalho desenvolvido, ou seja, são descritos os algoritmos de simplificação e refinamento de malhas desenvolvidos. Além disso são também apresentados e discutidos os resultados dos testes efectuados com alguns modelos de teste.
- No Capítulo 4 são apresentadas algumas conclusões sobre os resultados obtidos.

Capítulo 2

Trabalho Relacionado

2.1 Introdução

A utilização de malhas poligonais é cada vez maior, pois é necessário representar modelos cada vez com maior detalhe, os quais são normalmente aproximados por malhas triangulares para efeitos de manipulação e visualização. Este tipo de malhas são utilizadas em várias aplicações, tais como: aplicações de CAD, visualização, ambientes virtuais e jogos de computador.
Em aplicações como jogos de computador ou ambientes virtuais são muitas vezes utilizadas malhas com diferentes níveis de detalhe, uma vez que os objectos não necessitam de ter todos o mesmo nível de detalhe. Assim, começou a haver interesse nos algoritmos de simplificação e refinamento de malhas que permite criar malhas com diferentes níveis de detalhe [7, 8, 9]. Estes algoritmos permitem alterar o detalhe das malhas, reduzindo ou aumentando o número de polígonos. Para reduzir o número de polígonos de uma malha utiliza-se um algoritmo de simplificação, o que leva a que exista uma perda de qualidade da malha. A malha simplificada é utilizada quando esta está longe do utilizador, uma vez que nem sempre a perda de qualidade é perceptível ao olho humano, ou quando a malha é representada com um tamanho mais pequeno, logo não necessita de um nível de detalhe tão elevado como quando está mais próxima do utilizador [2]. Quando a malha está próxima do utilizador, ou seja, a malha é representada com um tamanho maior, logo o nível de detalhe tem de ser maior, pois a percepção dos detalhes da malha é maior do que se a malha estiver mais afastada do utilizador, para isso é normalmente utilizado um algoritmo de refinamento de malhas. Este tipo de métodos de simplificação e refinamento de malhas é geralmente utilizado com malhas de pequenas dimensões no que diz respeito a aplicações de tempo real, pois o processamento e troca entre níveis de detalhe é moroso e consome recursos. Para a manipulação de malhas existem duas soluções: os modelos LOD e os esquemas multi-resolução.
Em malhas de grandes dimensões são utilizados os modelos LOD, pois permitem a rápida troca entre níveis de detalhe. Por exemplo, a figura 2.1 apresenta três malhas de um coelho com diferentes resoluções. A malha com maior detalhe (malha mais à esquerda), é carregada quando

o modelo está mais próximo do utilizador, enquanto que as outras versões com menos detalhe são utilizadas quando a malha se afasta do utilizador, ou seja, quanto mais longe do utilizador estiver o modelo menos detalhe é necessário, logo é carregada uma das malhas com menor resolução [10, 11, 2]. Um dos problemas dos LOD é na transição entre níveis de resolução diferentes, pois pode notar-se a transição feita de uma resolução para a outra. Esta transição tem de ser feita de forma imperceptível ao utilizador para não haver uma mudança brusca entre a resolução que a malha tinha e a resolução que é pretendida nesse momento [12]. Outro problema dos LOD é que necessitam de mais memória para armazenar as diversas malhas com diferentes resoluções, isto é, várias malhas por modelo.

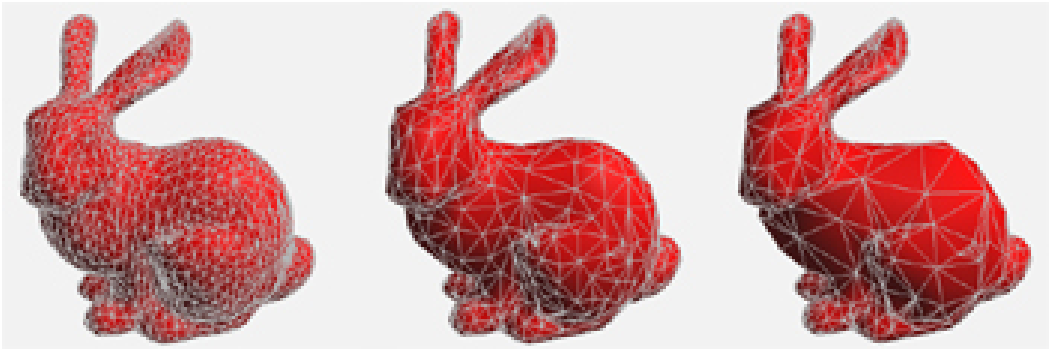


Figura 2.1: Malha de um coelho com três níveis de detalhe

Os esquemas multi-resolução são uma alternativa para ter modelos com vários níveis de detalhe. No entanto, estes não são utilizados para malhas de grandes dimensões em aplicações de tempo real, uma vez que a troca entre níveis de detalhe é feita de uma forma mais lenta do que com os LOD. Os esquemas multi-resolução têm em memória uma única instância do modelo, e para alternarem entre níveis de detalhe são necessários algoritmos de simplificação e refinamento de malhas [7, 4]. Como exemplo podemos ver na figura 2.1, onde a malha carregada inicialmente para a memória seria a malha mais à esquerda. As outras duas malhas seriam geradas quando a malha estivesse mais afastada do observador. Para isso é necessário um algoritmo de simplificação, enquanto que para a operação inversa é necessário um algoritmo de refinamento [13].

2.2 Esquemas Multi-Resolução

A grande vantagem e interesse dos esquemas multi-resolução em relação aos LOD é a possibilidade de alterar entre níveis de detalhe tendo só uma instância do modelo carregada em memória e permitir transições entre níveis de detalhe mais suaves.

2.2.1 Algoritmos de Simplificação

Os algoritmos de simplificação são usados em computação gráfica para criar malhas poligonais com menor detalhe.
Simplificar uma malha poligonal significa então, ter uma malha e através de um algoritmo de simplificação gerar uma malha com um menor número de células mas tentando manter a forma global do objecto. Os algoritmos de simplificação podem ser classificados dependendo do tipo de operação utilizada na simplificação da malha. As categorias são as seguintes: Remoção de células *(Cell Decimation)*, Confluência de vértices *(Vertex Clustering)* e Contracção de arestas *(Edge Collapsing)*.
Remoção de Células: Neste tipo de algoritmos é seleccionada uma célula (vértice, aresta ou face) para ser removida, depois desta ser removida são também removidas todas as células adjacentes, o que implica que exista uma reconstrução na zona onde foram removidas as células. Apesar de reduzir o número de células da malha a topologia do objecto é mantida. Pertencem a esta categoria os algoritmos descritos em [14, 15, 16, 17].
Confluência de Vértices: Este tipo de algoritmos envolvem a malha num volume e subdividem-no em sub-volumes mais pequenos. Esta subdivisão pode ser do tipo *octree*, que consiste em dividir o volume em oito sub-volumes. Isto é feito recursivamente até ser atingida a resolução adequada para o objecto. Dentro de cada sub-volume os vértices são depois representados por um vértice representativo *(cluster)*. Este vértice representativo pode ser o centro de massa do sub-volume ou qualquer outro ponto que seja escolhido por um critério pré-definido. Neste caso não é mantida a topologia do objecto. Alguns algoritmos que pertencem a esta categoria são [18, 19, 20, 21].
Contracção de Arestas: Os algoritmos deste tipo simplificam a malha através da contracção de arestas em vértices. O que diferencia os algoritmos desta família uns dos outros é a forma como seleccionam a aresta a contrair. Apesar das simplificações a forma global dos modelos é preservada na medida do possível. Algoritmos pertencentes a esta categoria são descritos em [10, 22, 1, 23].
Nestes algoritmos o que os distingue uns dos outros é o critério de selecção da aresta a contrair. Uma vez que se usam critérios diferentes para a selecção da aresta a contrair, os tempos de processamento e a qualidade das malhas também é diferente. De seguida são apresentados alguns algoritmos que recorrem à contracção de arestas para a simplificação de malhas.

No algoritmo apresentado por Garland e Heckbert [1] o critério para a selecção da aresta a contrair tem duas condições para que um par de vértices *(v1, v2)* seja válido para a contracção. A primeira condição é que o par de vértices *(v1, v2)* tem de pertencer à mesma aresta, e a segunda condição é que $\|v_1 - v_2\| < t$ em que t é um parâmetro de *threshold*. Este valor de *threshold* tem de ser cuidadosamente escolhido, pois pode ligar dois vértices que estejam muito afastados um do outro.
Para se localizar o conjunto válido de pares de vértices a contrair, cada vértice tem uma lista

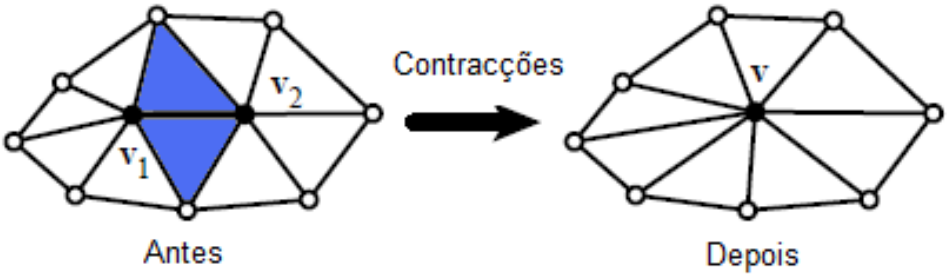


Figura 2.2: Contracção da aresta simples

dos quais é membro. Assim sendo, carrega-se uma matriz inicial com todos os vértices iniciais. A seguir, seleccionam-se todos os pares de vértices válidos e para cada par válido de vértices é calculado o custo de contracção desse par de vértices. Todos os pares são postos numa lista com o custo mínimo no inicio da lista. Iterativamente, remove-se o par de vértices *(v1, v2)* com o menor custo da lista, contrai-se esta aresta e actualiza-se o custo de todos os pares válidos envolvidos com o vértice *v1*. Se os dois vértices a contrair pertencerem à mesma aresta, uma ou mais faces são removidas (como mostra a figura 2.2). Caso contrário, as secções da malha envolvidas que estão separadas, juntam-se no vértice *v* (como mostra a figura 2.3).

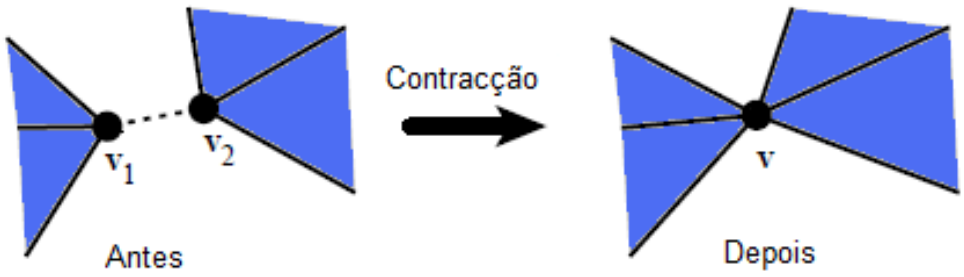


Figura 2.3: Contracção dos dois vértices quando não pertencem à mesma aresta (adaptado de [1])

No algoritmo proposto por Lindstrom e Turk [23], a selecção da aresta a contrair consiste em seleccionar repetidamente a aresta com o menor custo, colapsando essa aresta, e reavaliando o custo das arestas afectadas pela aresta colapsada, ou seja, segue uma estratégia similar à apresentada por Garland e Heckbert. Mas aqui, o custo de uma aresta é dado por uma função que encapsula a informação do volume e da área do modelo.
Em Guskov *et al.* [24], a simplificação das malhas é feita recorrendo ao algoritmo apresentado

por Garland e Heckbert, logo segue o mesmo critério.
No artigo de Hoppe [10], o critério para a selecção da aresta a contrair é feito através da minimização de uma função de energia. Uma malha progressiva é uma malha multi-resolução que pode ser simplificada através de contracções de arestas. A qualidade das malhas intermédias depende da forma como o algoritmo selecciona as arestas a contrair e quais os atributos que vão influenciar as arestas vizinhas. Para se encontrar a aresta a ser contraída é criada uma fila de prioridade em que para cada transformação é calculado o custo de energia. Em cada iteração, realiza-se uma transformação com o primeiro valor da fila, volta-se a calcular as prioridades das arestas vizinhas dessa transformação.
No algoritmo de simplificação proposto por Rodrigues *et al.* [4], foi utilizada uma estrutura de dados semelhante à estrutura de dados AIF, chamada de Estrutura de Dados baseada na Célula Fantasma (EDCF), em que a principal diferença é a existência de uma genealogia para os vértices e outra para as arestas, este algoritmo também é baseado na contracção da aresta. A genealogia do vértice serve para guardar a informação dos vértices antecessores dos vértices que ficam visíveis a cada simplificação. Esta informação contem a identidade dos dois vértices antecessores, P e Q, e armazena a informação relativa ao vector $v = \frac{Q-P}{2}$ para permitir recuperar a geometria dos vértices P e Q no processo de refinamento. A informação do vértice P é actualizada com os dados do novo vértice, enquanto que o vértice Q é colocado numa lista de vértices onde são guardados os vértices que são eliminados, para posteriormente poderem ser utilizados no refinamento.
A genealogia das arestas guarda a informação referente às arestas. Neste caso, é guardada a identificação dos vértices que fazem parte da aresta, *v1* e *v2*. A contracção da aresta é feita de modo a que *v1* e *v2* passam a ser o mesmo vértice, ou seja, a aresta fica reduzida a um único ponto, daí designar-se de *aresta fantasma*. Uma aresta só pode ser contraída se verificar as seguintes condições:

- a primeira, é que os vértices que pertencem à aresta sejam diferentes, ou seja, uma aresta contraída no actual nível de simplificação não pode ser contraída outra vez nesse nível.
- a segunda, é que a aresta não pode ser adjacente a uma aresta contraída no actual nível de simplificação.

O algoritmo de simplificação percorre a lista de arestas, e para cada aresta *(Vi, Vj)* que verifique o critério de contracção, esta é contraída da seguinte forma:

- Calcula-se o ponto médio *P(x, y, z)* da aresta *(Vi, Vj)*, ou seja, o ponto médio entre *Vi* e *Vj*
- Calcula-se então o vector de Hoppe através da seguinte expressão $\vec{v}_H = V_i - P$
- O vértice V_i é redefinido como um novo vértice V_{n+1} com as coordenadas calculadas anteriormente no ponto P, onde n é o identificador máximo dos vértices existentes.

- A informação genealógica de V_{n+1} e o vector $\vec{v}_H = V_i - P$ são guardados e é criada uma ligação com o vector Hoppe e a informação genealógica dos vértices que deram origem ao vértice V_{n+1}.
- Se existir informação genealógica associada aos vértices V_i e V_j, são definidas as sub-árvores genealógicas esquerda e direita do vértice V_{n+1}.
- Actualiza-se a lista de arestas da malha e a lista da arestas incidentes no vértice V_{n+1}, acrescentando a lista de arestas incidentes ao vértice V_j, evitando repetições de arestas.
- Por fim, insere-se V_j na lista de reciclagem de vértices da malha.

O algoritmo *Bi-Star Planarity (BSP)*, utiliza a contracção de arestas baseando-se na avaliação da complanaridade das faces em redor da aresta a ser contraída. O critério usado para escolher a aresta a contrair é puramente geométrico. A aresta a contrair é escolhida da seguinte forma:

- É seleccionada uma aresta ***e*** qualquer da lista de arestas;
- Se durante o mesmo passo de simplificação pelo menos um dos vértices da aresta ***e*** foi criado anteriormente por uma operação de contracção, então escolhe-se outra aresta;
- Se as faces que incidem em ***e*** não forem aproximadamente complanares dentro de uma tolerância ϵ, então é escolhida outra aresta;
- Se para cada vértice ***v*** da aresta ***e*** as faces incidentes em ***v*** não forem aproximadamente complanares de acordo com a tolerância ϵ, então é escolhida outra aresta;
- Para se contrair a aresta ***e*** num vértice, calcula-se o seu ponto médio, e em seguida removem-se as faces incidentes.

Na figura 2.4, é mostrado um exemplo de como funciona o algoritmo BSP. Neste exemplo, a contracção da aresta ***e*** inicia-se pela aglutinação das faces *f11* e *f12* numa só face *f1*, assim como as faces *f21* e *f22* na face *f2*, ou seja, as arestas e_1^* e e_2^* são removidas. Finalmente, a aresta ***e*** é contraída, num dos seus vértices. É calculado o ponto médio entre os vértices v e v^*. A aresta e é removida, assim como o vértice v^*.

No algoritmo *Normal-based Simplification Algorithm (NSA)*, uma aresta só pode ser contraída se a variação das normais às faces à volta da aresta a ser contraída estiver na tolerância de um valor ε. Este valor ε é um valor que limita o ângulo entre a normal à face actual e a nova normal à face depois de ser feita a contracção da aresta [3] , como mostra a figura 2.5. Quanto maior o valor de ε definido pelo utilizador, maior será a simplificação feita à malha.

Este critério geométrico implica que a região em redor da aresta a ser contraída tem de ser aproximadamente complanar. Mas o contrário não é verdade, uma vez que, a complanaridade não assegura que a variação das normais às faces exista. A figura 2.6 mostra dois casos de

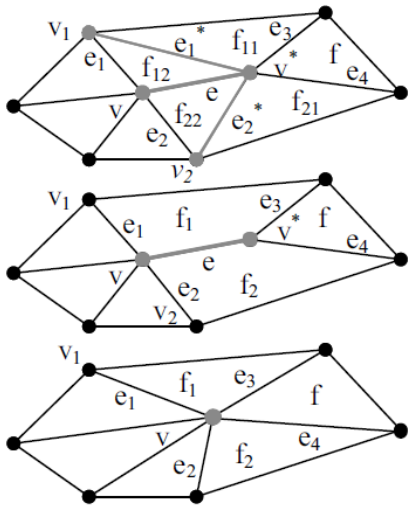


Figura 2.4: Simplificação da malha pelo algoritmo BSP (adaptado de [2])

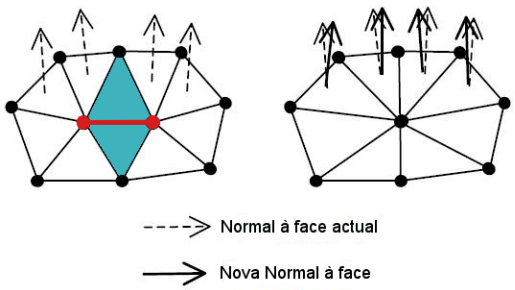


Figura 2.5: Normais às faces e suas variações (adaptado de [3])

faces complanares, o primeiro tem duas faces com normais com a mesma orientação, e o segundo tem as faces com normais com orientações contrárias.

2.2.2 Algoritmos de Refinamento

Os algoritmos de refinamento são outro tipo de algoritmos que se usam em computação gráfica para criar malhas poligonais com maior detalhe.
Refinar uma malha poligonal significa então, que a partir de um modelo grosseiro e através de um algoritmo de refinamento podemos gerar uma malha com um nível de detalhe maior, ou seja, por adição de novas células (vértices, arestas, faces) ao modelo. Para refinar uma malha é necessário que seja guardada alguma informação durante a simplificação. Por exemplo, na contracção de arestas são guardadas a aresta a contrair, o ponto médio da aresta e a distância de um dos vértices ao ponto médio. Esta informação, vai permitir que no refinamento se recupere

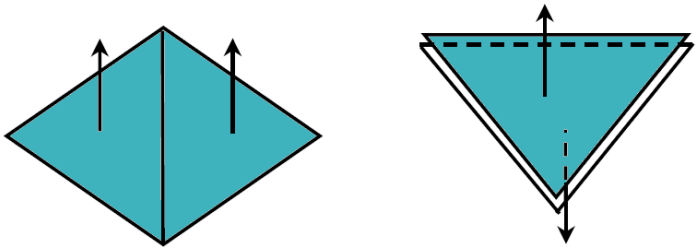


Figura 2.6: Exemplo de diferentes faces complanares (adaptado de [3])

a posição original dos vértices.
Por exemplo, na contracção da aresta **AB** da figura 2.7 iríamos guardar também o ponto médio (**M**) e a distância de um vértice ao ponto médio (**D/2**). Assim, com base nesta informação o algoritmo de refinamento pode recuperar os vértices **A** e **B** da seguinte forma:

$$\mathbf{A}=\mathbf{M}-\mathbf{D}/2$$
$$\mathbf{B}=\mathbf{M}+\mathbf{D}/2$$

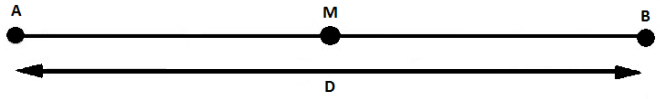


Figura 2.7: Onduleta de Harr

De um modo geral, os diversos algoritmos de refinamento associados aos algoritmos de simplificação, não são descritos em pormenor. No entanto, qualquer um deles baseia-se em informação guardada durante a simplificação para poder recuperar a malha original.
No algoritmo de refinamento proposto por Rodrigues *et al.* [4], a reconstrução da malha vai ser usada a informação guardada na simplificação, ou seja, é usada a árvore genealógica de cada vértice visível e a pilha genealógica de cada aresta que vai permitir reconstruir as relações topológicas entre as células da malha. O algoritmo de refinamento vai executar operações sobre a lista de vértices visíveis da malha. Para cada vértice V_n em que n é o identificador do vértice, o algoritmo vai executar as seguintes operações:

- Transfere-se da lista de reciclagem de vértices para a lista de vértices visíveis o primeiro vértice V_j.

- Restaura-se a identidade do vértice V_j partindo do identificador do nó direito da árvore

genealógica de V_n

- É restaurado também o identificador do vértice V_i a partir do identificador guardado no nó esquerdo da árvore genealógica.
- Partindo do vector de Hoppe guardado no vértice V_n restaura-se a geometria dos vértices V_i e V_j
- Restauram-se as árvores genealógicas de V_i e V_j partindo das subárvores direita e esquerda de V_i, ou seja o antigo V_n
- Actualizam-se as arestas que eram demarcadas pelo vértice V_n com os identificadores dos vértices restaurados V_i e V_j
- Finalmente actualizam-se as listas de arestas incidentes em V_i e V_j

2.2.3 Algoritmos de subdivisão

Os algoritmos de subdivisão são outro tipo de algoritmos que se usam em computação gráfica para criar malhas poligonais com maior detalhe. Neste caso, ao contrário do que acontece com os algoritmos de simplificação, os algoritmos de subdivisão partem de um modelo grosseiro e subdivide-se a malha para gerar um modelo mais detalhado.

Os algoritmos de subdivisão adicionam novas células aos modelos de forma a criarem um modelo com mais detalhe. Estas novas células são criadas inserindo novos vértices, subdividindo as arestas e/ou faces. Estas subdivisões podem ser feitas ou só nas arestas, ou nas faces, ou então numa combinação das duas formas [2, 25, 26]. Um exemplo disso, é a figura 2.8, onde os esquemas (a) e (d) representam subdivisões das arestas [27, 28, 29, 30, 31, 32] , os esquemas (c) e (f) representam subdivisões das faces e os esquemas (b) e (e) são uma combinação entre os dois esquemas já referidos [33, 34, 35, 36, 37]. A diferença entre os esquemas de subdivisão de malhas, consiste na forma como se calculam as coordenadas dos novos vértices.

Mais detalhes sobre os esquemas de subdivisão de malhas podem ser encontrados em [38].

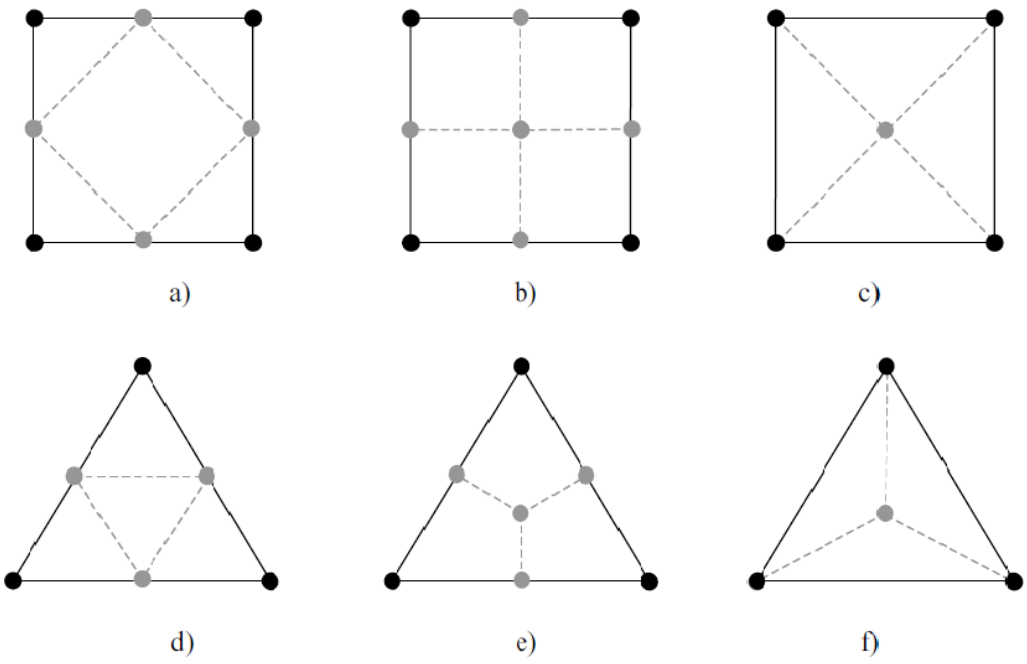


Figura 2.8: Tipos de subdivisão em faces triangulares e quadrangulares (retirada de [2])

Capítulo 3

Trabalho Desenvolvido

O trabalho desenvolvido nesta dissertação, teve como objectivo avaliar a utilidade de sistemas multi-resolução em sistemas de tempo real como, por exemplo, em sistemas de animação ou jogos. É sabido que os esquemas multi-resolução podem ser uma alternativa aos modelos com vários níveis de detalhe, mas certamente só em alguns casos. Neste sentido, pretendeu-se saber em que circunstâncias é possível usar um esquema multi-resolução e quando é preferível usar modelos com vários níveis de detalhe.
Para isso, foi implementado um esquema multi-resolução simples e avaliado o seu desempenho para diversos modelos com diferentes dimensões.
O esquema multi-resolução desenvolvido, baseia-se num algoritmo de simplificação e outro de refinamento de malhas, e permite criar quatro níveis de detalhe para cada modelo mantendo a forma global do modelo.
No decorrer deste capítulo, vão ser descritos o algoritmo de simplificação e o algoritmo de refinamento utilizados assim como, a estrutura de dados que serviu de suporte a este trabalho, bem como, os testes efectuados e resultados obtidos.

3.1 Estrutura de Dados AIF

A estrutura da dados AIF, foi escolhida para servir de suporte ao trabalho. Esta estrutura de dados, suporta malhas poligonais genéricas. As malhas podem ou não ser triangulares e podem ser *manifold* ou não. A estrutura de dados não é orientada, mas pode ser orientável por indução topológica.
Uma malha poligonal na estrutura de dados AIF, é representada por um conjunto de células (vértices, arestas e faces) com relações de adjacência e incidência entre si. Na figura 3.1, poderemos ver um exemplo de um objecto e o conjunto de células desse objecto, em que *(F1)* é a face, *(E1, E2, E3)* são as arestas e *(V1, V2, V3)* são os vértices do objecto [2, 39].
No exemplo da figura 3.1, a malha é definida como um conjunto de M= { *V, E, F* } , onde M

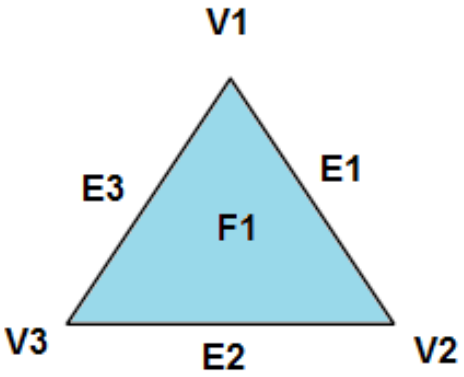


Figura 3.1: Exemplo de células num objecto

corresponde à malha propriamente dita, V é um conjunto finito de vértices, E é um conjunto finito de arestas e F é um conjunto finito de faces.
Um vértice $v \in V$, é definido por um conjunto de arestas incidentes em v, ou seja, $v = \{e_1, e_2, ..., e_k\}$.
A definição de uma aresta $e \in E$ é feita por dois conjuntos, em que o primeiro é referente aos vértices adjacentes à aresta, e o segundo é referente às faces incidentes nesta, ou seja, $e = \{\{v_1, v_2\}, \{f_1, f_2, ..., f_k\}\}$. Os vértices adjacentes à aresta são v_1 e v_2, e as faces incidentes na aresta são $f_1, f_2, ..., f_k$.
Uma face $f \in F$, define-se por um conjunto de arestas que lhe são adjacentes, por exemplo, na figura 3.1 a face F1 é descrita por $F1 = E_1, E_2, E_3$
Com estas relações de incidências e adjacências, podem representar-se malhas poligonais quer sejam *manifold* ou *non-manifold*.

3.1.1 Codificação - AIF

A estrutura de dados AIF representa malhas poligonais, sendo estas constituídas por um conjunto de células (vértices, arestas, faces). Cada uma das células é codificada através de uma classe em C++. A codificação de cada classe da estrutura de dados AIF é a seguinte:

```
class Point{
        double x,y,z; // coordenadas (x,y,z) da cada ponto
        }

class Vertex{
        int vid; // identificador correspondente ao vértice
        evector lei; // lista de arestas incidentes no vértice
```

```
        Point *pt; // geometria correspondente ao vértice
    }

class Edge{
        int eid; // identificador correspondente à aresta
        Vertex *v1, *v2; // vértices adjacentes à aresta
        fvector lfi; // lista de faces incidentes à aresta
        }

class Face{
        int fid; // identificador correspondente à face
        evector lei; // lista de arestas incidentes à face
        Point *nf; // normal à face
        }

class Mesh{
        int mid; // identificador correspondente à malha
        evector lvi; // lista de vértices da malha
        evector lei; // lista de arestas da malha
        evector lfi; // lista de faces da malha
        }
```

Mais detalhes sobre a estrutura da dados AIF, que serviu de suporte a este trabalho, podem ser encontrados em [2, 39].

3.2 Simplificação e Refinamento de Malhas

O objectivo de um algoritmo de simplificação é reduzir o número de células de uma malha poligonal, ou seja, reduzir o número de vértices, arestas e faces, mantendo a forma global do objecto.
O objectivo principal de um algoritmo de refinamento, é reconstruir uma malha a partir de uma malha mais simplificada. Neste caso, a reconstrução da malha faz-se com recurso à informação guardada durante a sua simplificação.
A seguir são descritos o algoritmo de simplificação e o algoritmo de refinamento utilizados neste trabalho, que serviram para implementar o esquema multi-resolução usado neste trabalho.

3.2.1 Algoritmo de Simplificação

O algoritmo desenvolvido baseia-se na operação de *contracção da aresta (ver figura 3.3)*. Esta operação, consiste em contrair a aresta num único vértice. Neste algoritmo, a aresta não vai

ser eliminada, simplesmente fica com comprimento nulo, ou seja, os dois vértices que compõem a aresta ficam sobrepostos um ao outro dando a sensação que a aresta foi eliminada e que foi substituída por um vértice.
Cada aresta tem uma *flag* (*ON* ou *OFF*), que indica se a aresta já foi contraída ou não.
O algoritmo vai percorrer a lista de arestas da malha, e para cada aresta e_i em que a *flag* esteja marcada como *OFF*, esta é contraída do seguinte modo:

- Marcam-se as arestas adjacentes aos vértices V_i e V_j como *ON*, para que a contracção das arestas não seja só feita numa zona da malha, obrigando assim a percorrer a malha toda. Tomando como exemplo a figura 3.3 as arestas que seriam marcadas como *ON* seriam, e_1, e_2, e_3, e_4, e_5 pois são arestas adjacentes ao vértice V_i, as arestas $e_6, e_7, e_8, e_9, e_{10}$ também seriam marcadas pois são adjacentes ao vértice V_j. Na figura 3.3 é mostrada a aresta a ser contraída enquanto as arestas adjacentes aos vértices que a compõem são marcadas a *ON* para não serem utilizadas nas próximas iterações de modo a que o algoritmo percorra todas as arestas da malha. As arestas que não estão marcadas como *ON* poderão ser utilizadas na próxima iteração, e repete-se estes passos até que o algoritmo tenha percorrido a malha na sua totalidade. Quando o algoritmo acabar de percorrer todas as arestas da malha, as arestas que não foram contraídas são marcadas novamente como *OFF* para que possam ser utilizadas na próxima simplificação.
- Calcula-se o ponto médio dos vértices V_i e V_j, usando a seguinte equação $PM = \frac{V_i + V_j}{2}$ (ver figura 3.2).
- Calcula-se a distância D entre o vértice V_i e o ponto médio previamente calculado. Esta distância é necessária para recalcular a posição original dos vértices quando for feito o refinamento da malha. Esta distância D, é dada pela seguinte equação $D = V_i - PM$. Como se pode ver pela figura 3.2 o ponto médio de V_i e V_j é PM e a distância D é metade da aresta que está a ser tratada no momento.
- Actualiza-se as coordenadas de V_i e V_j com as coordenadas do ponto médio, PM. É guardada uma lista dos identificadores dos vértices que estão sobrepostos a V_i e V_j, ou seja, as arestas que têm comprimento nulo.
- A informação da distância, ponto médio, nível de simplificação e a lista dos identificadores dos vértices sobrepostos a V_i e V_j são guardadas na classe aresta, para poderem ser acedidas quando se refinar a malha.

Quando não existem mais arestas a *OFF*, termina uma simplificação da malha (isto corresponde a um nível de simplificação). Depois, é percorrida novamente a lista de arestas e são colocadas a *OFF* todas as arestas que não tenham comprimento nulo. Desta forma, é possível efectuar uma nova simplificação à malha, ou seja, pode-se aplicar mais uma simplificação à malha (isto é um aumento no nível de simplificação).

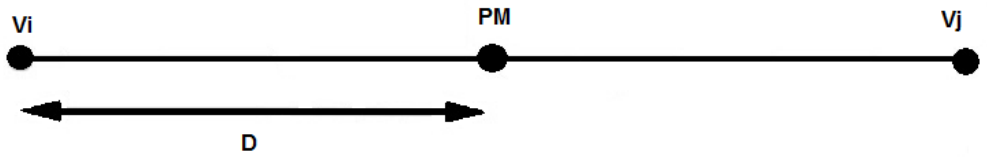


Figura 3.2: Ponto médio e distância de uma aresta

3.2.2 Algoritmo de Refinamento

Este algoritmo baseia-se na operação de *divisão do vértice (ver figura 3.3)*. Esta operação, consiste em dividir um vértice em dois e unir estes dois vértices por uma aresta. Neste caso, a aresta já existe uma vez que na simplificação ela não é eliminada, mas simplesmente os vértices ficam sobrepostos um ao outro, ficando assim a aresta com tamanho nulo, dando a sensação que existe somente um vértice.
O algoritmo de refinamento percorre a lista de arestas, e para cada aresta e_i em que a flag de controlo é *ON* e está no nível actual de simplificação, este acede à informação guardada na simplificação, ou seja, a distância D, o ponto médio PM e a lista de vértices sobrepostos a V_i e V_j. Com esta informação é possível restaurar a malha. Assim o algoritmo faz:

- Restauração da posição inicial dos vértices V_i e V_j. Para restaurar a posição inicial do vértice V_i, é necessário somar ao ponto médio PM a distância D, ou seja, $V_j = PM + D$. Enquanto, que para restaurar a posição inicial do vértice V_i é necessário subtrair ao ponto médio PM a distância D, ou seja, $V_i = PM - D$.

- Aceder à lista onde foram guardados os vértices que estão sobrepostos a V_i e V_j, e actualizar adequadamente a sua posição.

- Alterar o estado da *flag* de controlo da aresta de *ON* para *OFF*, e alterar o nível da aresta do nível actual para o nível anterior, ou seja, decrementar o nível de simplificação.

Tomando como exemplo a figura 3.3, a imagem da esquerda mostra uma aresta que foi colapsada, ou seja, os dois vértices correspondentes da aresta estão sobrepostos ($V_i == V_j$). Para restaurar as posições originais dos dois vértices, é necessário subtrair/adicionar o valor da distância a cada um dos vértices.

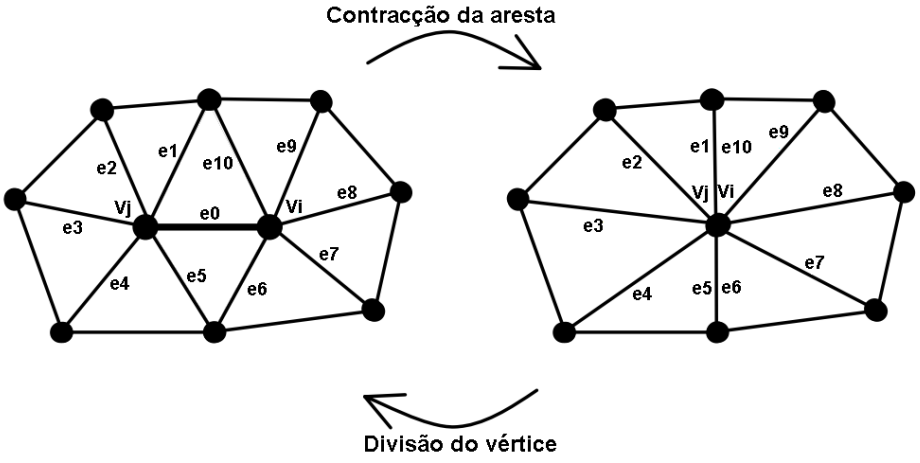


Figura 3.3: Contracção da aresta e divisão do vértice

3.3 Testes e Resultados

Os testes foram efectuados num computador desktop com um processador Intel(R) Core(TM)2 Quad CPU Q6600 a 2.40GHz, com 4GB de RAM, com o sistema operativo Windows XP Professional de 32 bits © e uma placa gráfica NVIDIA GeForce 8800 GT com 512 MB de memória.

3.3.1 Modelos utilizados e testes

Para a realização dos testes, foi necessário implementar também algumas funcionalidades para facilitar a manipulação das malhas. As funcionalidades implementadas foram as seguintes: carregar malhas no formato *Object File Format (OFF)*, seleccionar malhas, e as operações geométricas tradicionais, translações, rotações e variação de escala para a malha seleccionada. A figura 3.4, mostra os quatro modelos utilizados nos testes, o oito, a cabeça de dragão, a coruja e a vaca. No máximo foram considerados cinco níveis de detalhe, como se ilustra na tabela 3.1. Significa, que para além do modelo original, que está associado ao nível zero na tabela 3.1, é possível criar mais quatro simplificações. Assim, dependendo da distância da malha ao observador esta tem menos ou mais detalhe.
Para cada um dos modelos considerados, foi calculado o tempo de simplificação e refinamento que é necessário para a passagem entre os vários níveis de detalhe.

Na tabela 3.1, é apresentado o número de faces de cada uma das malhas para os cinco níveis de detalhe considerados.

Nos gráficos da figura 3.5, são apresentados os tempos de simplificação e de refinamento para cada uma das malhas em segundos. Como se pode ver pelos gráficos, os tempos de simplificação

(a) Dragão (b) Vaca (c) Oito (d) Coruja

Figura 3.4: Malhas utilizadas nos testes

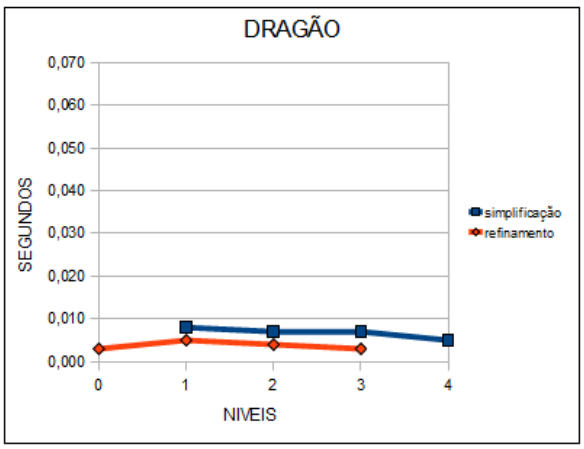
Malhas	dragão	oito	coruja	vaca
Nível 0	584	1536	3548	5804
Nível 1	250	804	1832	2964
Nível 2	150	382	960	1154
Nível 3	91	152	426	499
Nível 4	58	64	144	174

Tabela 3.1: Número de faces para cada malha por nível de detalhe

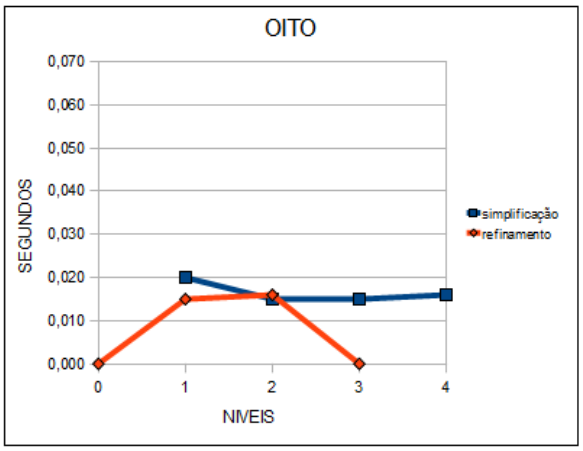
e refinamento são menores para malhas com menor número de faces (3.5(a) e 3.5(b)), e são maiores para malhas com maior número de faces (ex:3.5(c) e 3.5(d)). O tempo de simplificação e/ou refinamento depende do número de células do modelo, ou seja, quanto maior o número de vértices, faces e arestas do modelo, maior será o tempo necessário para simplificar e para refinar.

Nos gráficos da figura 3.6, é apresentado o número de *Frames Por Segundo (FPS)* para cada uma das malhas, quando é feita a simplificação e o refinamento das mesmas para os diversos níveis de detalhe. Como se pode constatar, para malhas de dimensão inferior a 2000 faces é possível obter um número de FPS acima de 60. Isto significa, que pode ser usado um esquema multi-resolução para obter vários níveis de detalhe em sistemas de tempo real. Para malhas de dimensão superior a 3000 faces, o número de FPS já desce abaixo dos 30 FPS. Logo, para malhas desta dimensão já é mais difícil usar um esquema multi-resolução num sistema de tempo real. No entanto, como os sistemas podem ter de gerir em simultâneo mais do que uma malha, foram efectuados mais alguns testes agrupando as malhas duas a duas.

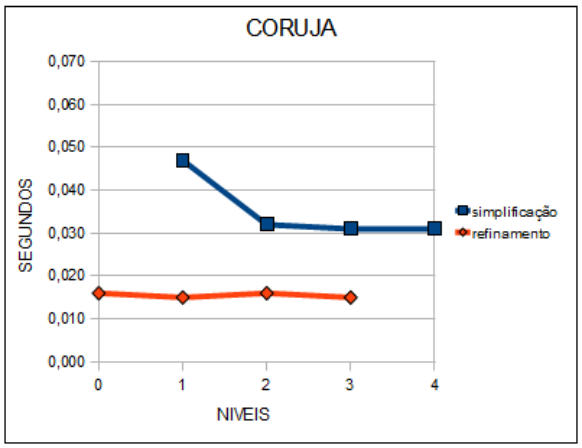
Assim, foram escolhidos os pares (Oito, Dragão), (Dragão, Coruja), (Vaca, Oito), e (Coruja, Vaca), pois permite verificar o comportamento dos modelos quando o número de células das malhas é menor que 2200, 4200, 6400 e 9400 respectivamente. Para estes testes, carregaram-se os modelos em posições diferentes, pois num sistema de tempo real as malhas podem encontrar-se em localizações diferentes no espaço, como exemplo, temos a figura 3.7 em que foram carregadas as malhas da Vaca e da Coruja respectivamente. Para cada par de malhas, foi calculado o tempo de simplificação e de refinamento de cada par de malhas e os respectivos FPS. Note-se, que



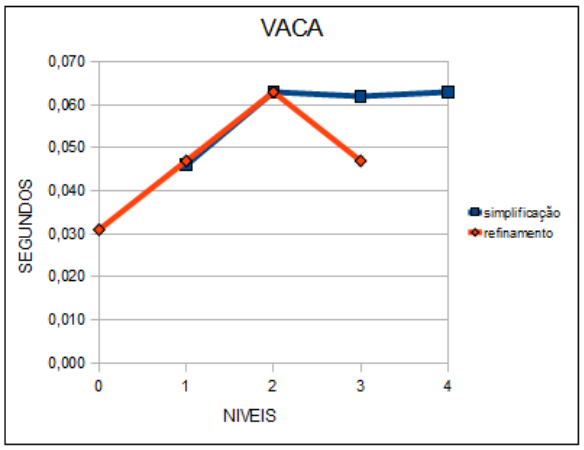
(a) Malha do Dragão



(b) Malha do Oito



(c) Malha da Coruja



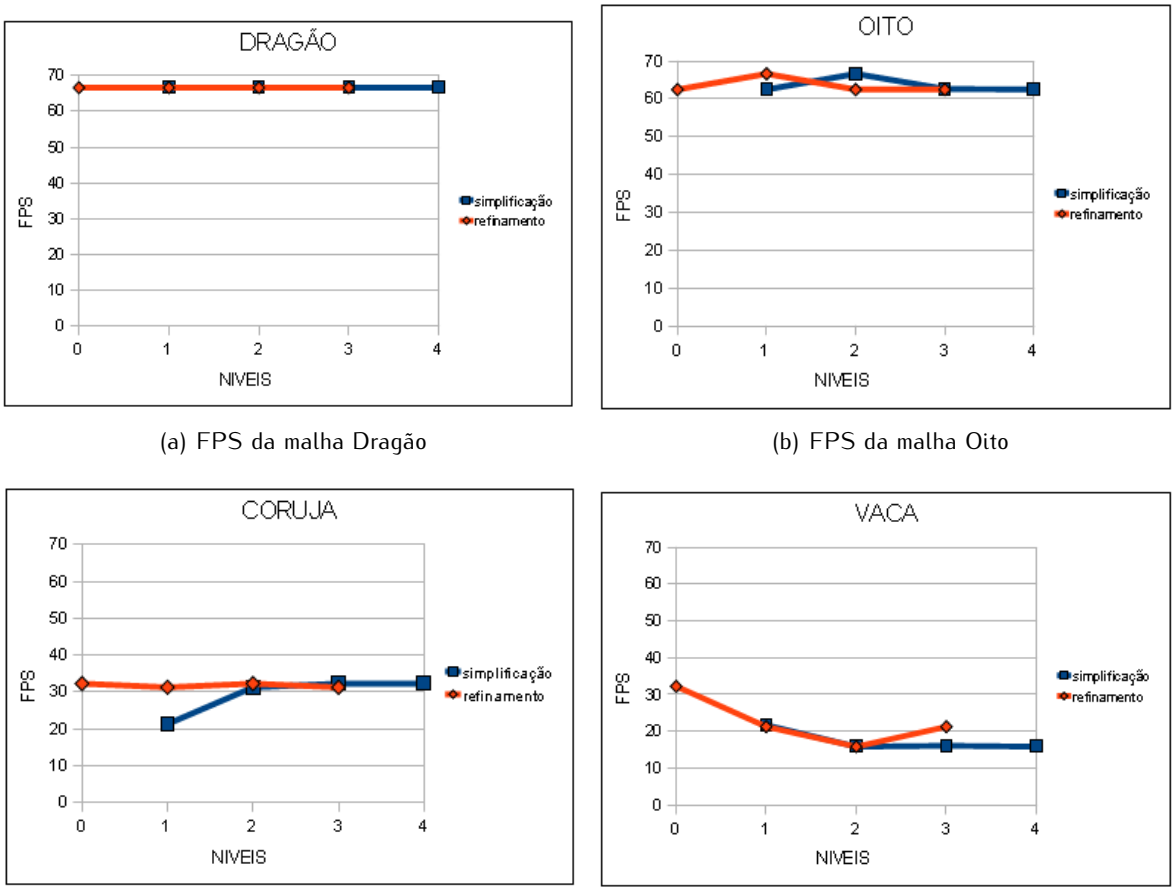
(d) Malha da Vaca

Figura 3.5: Tempos de simplificação e refinamento para cada malha

foi simulado que quando uma malha está a aproximar-se do utilizador a outra está a afastar-se. Logo, enquanto uma malha é simplificada a outra é refinada.

Na figura 3.8, são apresentados os tempos em segundos da simplificação e refinamento dos pares de malhas considerados. Podemos verificar, que as malhas com tamanhos mais pequenos o comportamento na simplificação e refinamento é relativamente semelhante (3.8(c)), o mesmo não se aplica a malhas com tamanhos maiores (3.8(d)). As variações dos tempos na simplificação e refinamento, são mais notórias quando se juntam malhas grandes (3.8(d)) uma vez que o número de células que é necessário processar é maior o que torna a sua simplificação e o seu refinamento mais moroso.

A figura 3.9, apresenta os FPS para cada um dos pares de malhas considerados. Como se pode verificar, pode obter-se um número de FPS acima dos 60 para malhas em que o número total de células é inferior a 2200 faces 3.9(c). Isto significa, que pode utilizar-se um esquema de



(a) FPS da malha Dragão

(b) FPS da malha Oito

(c) FPS da malha Coruja

(d) FPS da malha Vaca

Figura 3.6: FPS na simplificação/refinamento para cada malha

multi-resolução para obter vários níveis de detalhe em sistemas de tempo real que manipulem malhas destas dimensões (ex: jogos). Para malhas com um número de faces superior a 4000, o número de FPS desce abaixo dos 30, o que torna mais difícil a utilização de um esquema multi-resolução num sistema de tempo real.

Resumindo, a utilização de um esquema multi-resolução só será viável num sistema em tempo real se as malhas a manipular tiverem dimensões não muito grandes (ex: até 2000 faces). Caso contrário, poderá afectar a performance do sistema e por isso é melhor usar modelos com vários níveis de detalhe.

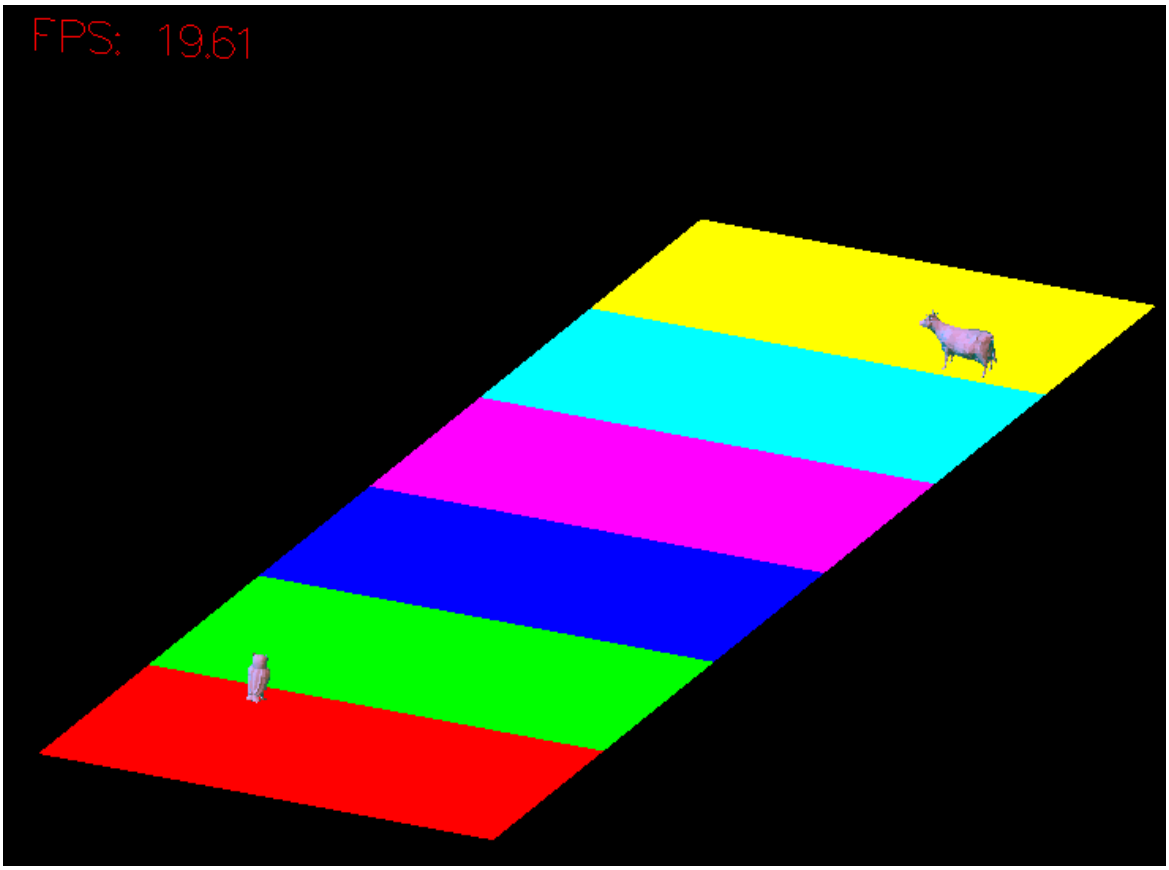
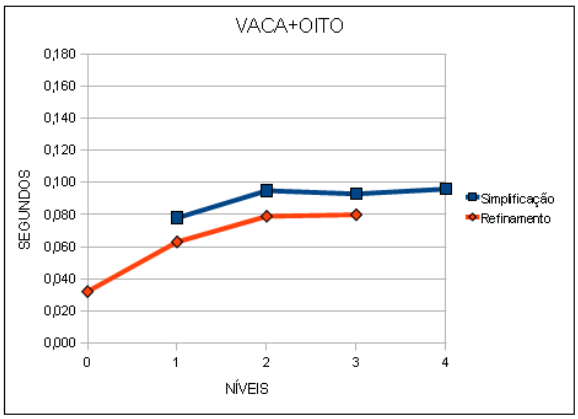
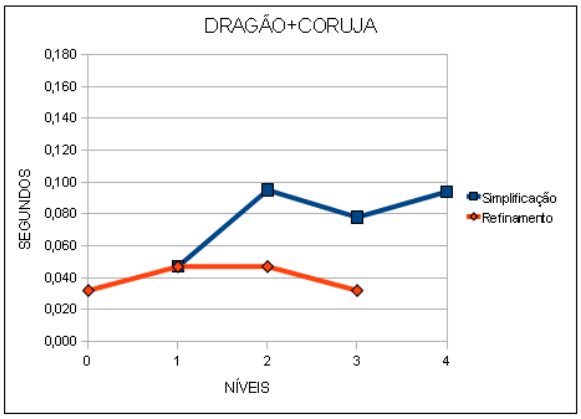


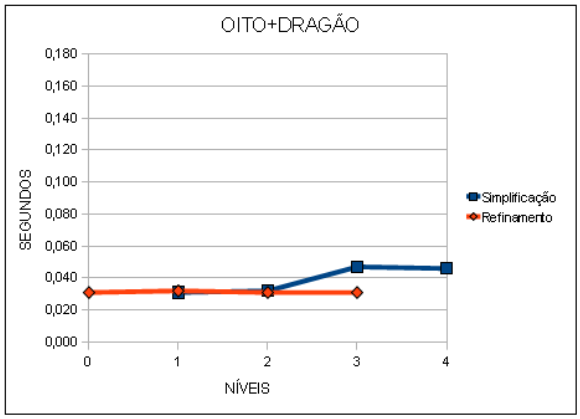
Figura 3.7: Exemplo de duas malhas carregadas



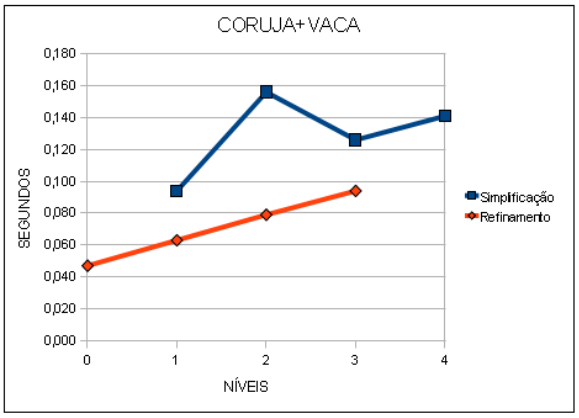
(a) Malhas Vaca e Oito



(b) Malhas Dragão e Coruja

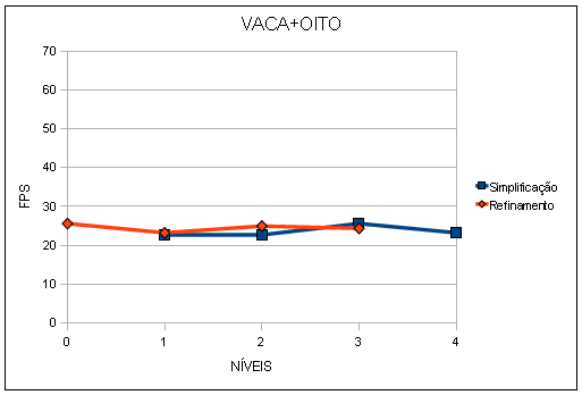


(c) Malhas Oito e Dragão

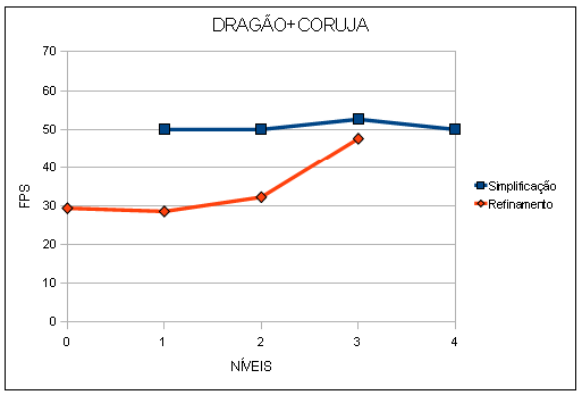


(d) Malhas Coruja e Vaca

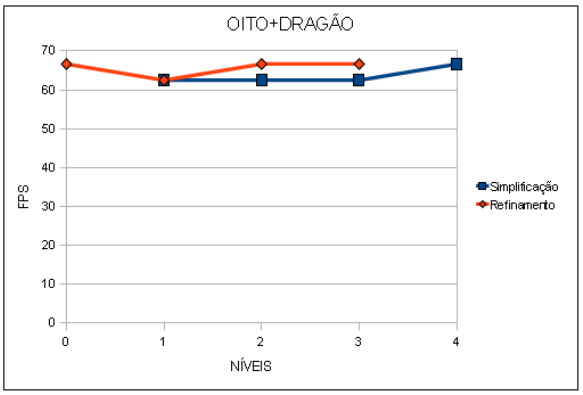
Figura 3.8: Tempos de simplificação e refinamento dos pares de malhas considerados



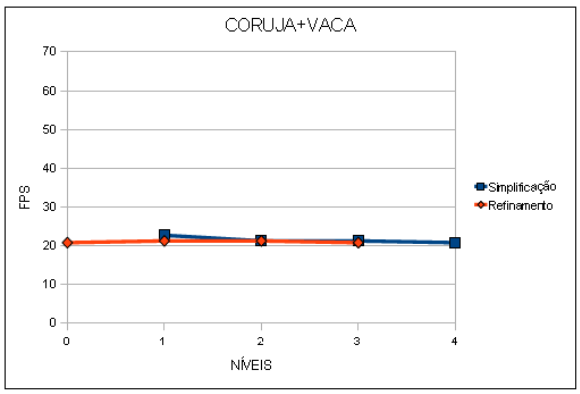
(a) FPS das malhas Vaca e Oito



(b) FPS das malhas Dragão e Coruja



(c) FPS das malhas Oito e Dragão



(d) FPS das malhas Coruja e Vaca

Figura 3.9: FPS da simplificação/refinamento dos pares de malhas considerados

Capítulo 4

Conclusões

O principal objectivo deste trabalho, foi avaliar a análise multi-resolução para representar modelos em sistemas de tempo real como por exemplo, jogos de computador ou sistemas de animação. Para tal, foram efectuados alguns testes e avaliados os resultados com um esquema multi-resolução simples desenvolvido para malhas poligonais. Como se pôde ver no capítulo anterior, o tempo necessário para efectuar a simplificação/refinamento de malhas depende do número de células que têm de ser processadas em cada instante, ou seja, quanto maior for o número de células de cada malha e maior o número de malhas carregadas no ecrã, maior será o tempo necessário para efectuar a respectiva simplificação/refinamento. Para além dos tempos de simplificação e de refinamento foi ainda avaliado o número de *frames* por segundo que se conseguia atingir com a utilização do esquema multi-resolução testado. Assim, foram calculados os FPS para a simplificação e o refinamento das malhas usadas no teste, bem como, para conjuntos de duas malhas. Os resultados, mostram que para malhas até 4000 faces é possível usar um esquema multi-resolução para obter vários níveis de detalhe, mesmo em aplicações de tempo real. Já para malhas de maiores dimensões, será preferível recorrer a modelos com vários níveis de detalhe. Ou seja, para malhas com um número de faces superior a 4000, torna-se difícil a utilização de um esquema multi-resolução uma vez que, o número de FPS desce abaixo dos 30. Para malhas com um número de faces inferior a 4000, a utilização de um esquema multi-resolução já é possível.
Logo, quando o número de malhas e células é grande, continua a ser preferível utilizar os LOD uma vez que, apesar de consumir mais memória do que os esquemas multi-resolução, é mais rápido na troca de níveis de detalhe das malhas.

Referências

[1] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216. ACM Press/Addison-Wesley Publishing Co., 1997.

[2] Frutuoso G. M. Silva. *Estruturas Poligonais Deformáveis com Resolução Variável*. PhD thesis, Universidade da Beira Interior, 2005.

[3] Frutuoso G. M. Silva and Abel J. P. Gomes. Normal-based simplification algorithm for meshes. *IEEE*, 2004.

[4] Rui Rodrigues, José Morgado, Frutuoso Silva, and Abel Gomes. A ghost cell-based data structure for multiresolution meshes. *IEEE*, 2007.

[5] Jinseok Seo, Gerard Jounghyun Kim, and Kyo Chul Kang. Levels of detail (lod) engineering of vr objects. In *VRST '99: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 104–110, New York, NY, USA, 1999. ACM.

[6] Benjamín Hernandez and Isaac Rudomin. Simple dynamic lod for geometry images. *ACM*, 2006.

[7] M. Garland. Multiresolution modeling: Survey & future opportunities. *IEEE*, 1999.

[8] Taosong He, Lichan Hong, Amitabh Varshney, and Sidney Wang. Controlled topology simplification. *IEEE*, 1996.

[9] Francisco Lopez, Ramon Molla, and Veronica Sundstedt. Exploring peripheral lod change detections during interactive gaming tasks. In *APGV '10: Proceedings of the 7th Symposium on Applied Perception in Graphics and Visualization*, pages 73–80, New York, NY, USA, 2010. ACM.

[10] Hugues Hoppe. Progressive meshes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108, New York, NY, USA, 1996. ACM.

[11] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. *ACM*, 1995.

[12] Derrick Parkhurst and Ernst Niebur. A feasibility test for perceptually adaptive level of detail rendering on desktop systems. In *APGV '04: Proceedings of the 1st Symposium on Applied perception in graphics and visualization*, pages 49–56, New York, NY, USA, 2004. ACM.

[13] Scott Kircher and Michael Garland. Progressive multiresolution meshes for deforming surfaces. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 191–200, New York, NY, USA, 2005. ACM.

[14] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 65–70. ACM, 1992.

[15] Reinhard Klein, Gunther Liebich, and W. Straßer. Mesh reduction with error control. *ACM*, 1996.

[16] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, and William Wright. Simpli?cation envelopes. *ACM*, 1996.

[17] Leif Kobbelt, Swen Campagna, and Hans-Peter Seidel. A general framework for mesh decimation. In *Graphics Interface*, pages 43–50, 1998.

[18] Peter Lindstrom. Out-of-core simplification of large polygonal models. In *Proceedings of SIGGRAPH 2000*, pages 259–262, 2000.

[19] Nor Anita Fairos bt. Ismail, Mohd Shafry Mohd Rahim, Daut Daman, and Sheikh Nasir Kamarudin. Out of core simplification with appearance preservation for computer game applications. *IEEE*, 2009.

[20] Dmitry Brodsky and Benjamin Watson. Model simplification through refinement. In *PROC. OF GRAPHICS INTERFACE*, pages 221–228, 2000.

[21] Jarek Rossignac and Paul Borrel. Multi-resolution 3d approximations for rendering complex scenes. *In Geometric Modeling in Computer Graphics*, 1993.

[22] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 19–26, New York, NY, USA, 1993. ACM.

[23] Peter Lindstrom and Greg Turk. Fast and memory efficient polygonal simplification. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 279–286, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[24] Igor Guskov, Kiril Vidimče, Wim Sweldens, and Peter Schröder. Normal meshes. *ACM*, 2000.

[25] MICHAEL LOUNSBERY, TONY D. DeROSE, and JOE WARREN. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Transactions on Graphics*, 1997.

[26] Norbert Pfeifer. A subdivision algorithm for smooth 3d terrain models. *ISPRS Journal of Photogrammetry & Remote Sensing*, 2005.

[27] Nira Dyn, David Levine, and John A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Trans. Graph.*, 9(2):160–169, 1990.

[28] Denis Zorin and Peter Schröder. Subdivision for Modeling and Animation. Technical report, SIGGRAPH 2000, 2000. Course Notes.

[29] Charles Teorell Loop. Smooth subdivision surfaces based on triangles. Department of mathematics, University of Utah, Utah, USA, 1987.

[30] Leif Kobbelt. Interpolatory subdivision on open quadrilateral nets with arbitrary topology. In *Computer Graphics Forum*, pages 409–420, 1996.

[31] Joe Warren and Henrik Weimer. *Subdivision Methods for Geometric Design: A Constructive Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[32] S. Schaefer, J. Hakenberg, and J. Warren. Smooth subdivision of tetrahedral meshes. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 147–154, New York, NY, USA, 2004. ACM.

[33] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. pages 183–188, 1998.

[34] Tianyun Ni, Ignacio Casta no, Jörg Peters, Jason Mitchell, Philip Schneider, and Vivek Verma. Efficient substitutes for subdivision surfaces. In *SIGGRAPH '09: ACM SIGGRAPH 2009 Courses*, pages 1–107, New York, NY, USA, 2009. ACM.

[35] Chandrajit Bajaj, Scott Schaefer, Joe Warren, and Guoliang Xu. A subdivision scheme for hexahedral meshes. *The Visual Computer*, 18:343–356, 2002.

[36] Martin Bertram. Biorthogonal wavelets for subdivision volumes. In *In: Proceedings of the Seventh ACM Symposium on Solid Modeling and Applications*, pages 72–82, 2002.

[37] Luiz Velho. Stellar subdivision grammars. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 188–199, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[38] Malcolm Sabin. Recent progress in subdivision: a survey. In *In Advanced in Multiresolution for Geometric Modelling*, pages 203–230. Springer, 2004.

[39] Frutuoso G. M. Silva and Abel J. P. Gomes. Adjacency and incidence framework: a data structure for efficient and fast management of multiresolution meshes. In *GRAPHITE '03: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 159–166, New York, NY, USA, 2003. ACM.